Christian-Albrechts-Universität zu Kiel
Institut für Informatik und Praktische Mathematik
Prof. Dr. W.-P. de Roever
Harald Fecher

Sys. Inf. III (Betriebssysteme)

Wintersemester 2004/05 — **Klausur** — 29. Januar 2005, 10:00–13:00

Termin: 29. Januar 2005, 10:00–13:00

Um die Korrektur zu vereinfachen und zu beschleunigen:

- Geben Sie auf jedem Blatt Ihren **Namen**/Matrikelnummer an.
- Fangen Sie jede Aufgabe auf einem **neuen Blatt** an (nicht nur auf einer neuen Seite!) und kennzeichnen Sie das Blatt entsprechend!
- Die Verwendung von Unterlagen, einschließlich "dem Comer" ist erlaubt. Elektronische Unterlagen wie Handy und Computer sind verboten. Tachenrechner sind erlaubt.

Für Aufgaben, die in Xinu eingreifen, sollen auch Xinu-Bezeichner verwendet werden. Wenn Sie beispielsweise die Prozeßtabelle verändern wollen, verwenden Sie im Code den Bezeichner `proctab` etc. Falls die Aufgabe lautet „Ändern Sie das Verhalten der Prozedur, so daß. . . " und Sie die Aufgabe lösen, indem Sie vorhandenen Code teilweise wiederverwenden, müssen die Stellen, an denen ersetzt wird, *eindeutig* zu erkennen sein (zumindest muß Prozedurname mit Argumenten angegeben sein, auch Seitenzahl im Comer). Es ist nicht verlangt, irgendwelche `include`-Befehle anzugeben.

Für "abstrakte" Synchronisationsaufgaben (hier Aufgabe 2) ist es nicht verlangt, Xinu-spezifischen Code zu geben; in der Tat sollten konkrete hardwarenahe Eingriffe ("Interrupts ausschalten") nicht verwendet werden, da sie nicht zum Repertoire des *Benutzers* des Betriebssystems gehören. In jedem Fall ist neben sinnvoll kommentiertem Code eine *Beschreibung* Ihrer Lösung in einigen Sätzen zu geben.

Viel Erfolg!

Name:
Matrikelnummer:

Aufg 1	Aufg 2	Aufg 3	Aufg 4	Aufg 5	Gesamt	Note

Aufgabe 1 (Test & Set (5 Punkte)) Wir haben in der Vorlesung eingehend über das Problem des gegenseitigen Ausschlusses und mögliche Lösungen —mittels Software oder durch Manipulation der Interrupts— gesprochen. Es gibt auch *hardwaregestützte* Lösungen, gegenseitigen Ausschluß zu garantieren. Ein bekannte Variante aus dieser Kategorie wird als *Test & Set* bezeichnet und ist Gegenstand dieser Aufgabe.

Gegeben seien 2 Prozesse P_0, P_1 und eine globale Variable C, mit möglichen Werten 0 und 1 und mit initialem Wert 0. Die *test-and-set*-Operation greift auf diese globale Variable zu, kopiert ihren Wert in eine lokale Variable und setzt C "gleichzeitig" auf 1. Der Aufruf `test_and_set(L)` entspricht somit der nicht-unterbrechbaren, atomaren Ausführung der beiden Zuweisungen:

```
L := C;
C := 1;
```

Betrachten Sie den angegebenen Code für gegenseitigen Ausschluß mittels *test-and-set.*

```
/*       Test & set                                                    */
1)    process P_i                        /* der i-te Prozess          */

      int L_i;                           /* lokale Kopie fuer P_i     */
2)      loop                             /* Endlosschleife            */

3)        unkritischer_Code;

4)        loop                           /* innere Schleife           */
5)          test_and_set(L_i);
6)          exit when (L_i == 0);        /* Abbruch der inneren Schleife */
7)        end loop;

8)        Beginn_kritischer_Abschnitt_i;
9)        Ende_kritischer_Abschnitt_i;
10)       C := 0;
11)     end loop;
12) end process;
```

Beweisen Sie, daß die Implementierung den gegenseitigen Ausschluß garantiert, d.h., sie verhindert, daß sich zwei unterschiedliche Prozesse gleichzeitig in ihrem kritischen Abschnitt befinden.
Tipp: Überlegen Sie sich was gilt wenn folgende Aussage erfüllt ist:
(P_1 in CS $\vee$ (P_1 in Zeile 6 $\wedge L_1 = 0$))

Zeigen Sie, auf welche Weise der gegenseitige Ausschluß scheitert, wenn die *test-and-set*-Operation *nicht atomar* ist, sondern durch das angegebene Zuweisungspaar implementiert würde. (Die Ziffern im Code sind „Zeilennummern", die Sie in Ihren Argumenten verwenden können.)

Aufgabe 2 (Cocktail Bar (5 Punkte)) Betrachten Sie folgende abstrakte Beschreibung eines Synchronisations/Koordinationsproblems:

> Gegeben sei eine Bar mit zwei Barmännern. Beide zapfen Bier und Limonade. Drei Kunden mit unterschiedlichen Geschmäckern wollen bedient werden, der eine will Bier, der andere Limo, der dritte Radler.[1]
>
> Die Bedienung und das Trinken funktioniert *rundenweise.* Ein Getränk besteht aus 2 Einheiten. Jeder der Barleute zapft eine Einheit davon, und zwar zufällig/nichtdeterministisch Bier oder Limo. Je nachdem, was zusammengezapft wurde, trinkt der passende der drei Kunden. Ist das Getränk konsumiert, geht es in die nächste Runde, und immer so weiter. Halbe Getränke werden nicht konsumiert, neu ausgeschenkt wird nur, wenn das Getränk ganz konsumiert ist.

Modellieren Sie dieses Problem durch geeignete Prozesse, wobei jeder der drei Kunden und jeder der 2 Barmänner ein Prozeß sein soll. Der momentane Zustand des Getränkes sei durch das Variablenpaar `g1` und `g2` gegeben, wobei jeder der Variablen den Wert `BIER`, `LIMO`, und `LEER` annehmen kann (zu Beginn: beide `LEER`). Beispielsweise gibt

```
g1 = BIER
g2 = LIMO
```

den Zustand an, bei der jeder der beiden Barmänner ein Element produziert hat (entspricht einem fertigen Glas Radler). Die drei Kunden können lesend und schreibend auf diese gemeinsamen Variablen zugreifen; was die zwei Barmänner `Bar1` und `Bar2` betrifft, kann `Bar1` auf die Variable `g2` *nicht schreibend* zugreifen und `Bar2` entsprechend nicht auf `g1`.

Ein Ausschnitt des Radlertrinker-Prozeß könnte z.B. wie folgt aussehen:

```
Process Radlertrinker (){
  while (true) {

    .....              // Koordinationskode

    g1:=LEER;          // konsumiere Radler durch
    g2:=LEER;          // Ruecksetzen beider var's

    ...               // Koordinationskode

    }
  }
```

Dabei soll sichergestellt werden, daß er nur trinken darf, wenn vorher `g1` und `g2` auf `LIMO` resp. `BIER` gesetzt waren (oder umgekehrt). Was die Barleute betrifft: sie sollen erst wieder erneut anfangen "einzuschenken", d.h., in die Variablen zu schreiben —`Bar1` in `g1` und `Bar2`

[1] Für die Norddeutschen: auch als Alsterwasser bekannt = Bier + Limonade.

in `g2`— nachdem einer der Kunden beide Variablen auf `LEER` gesetzt hat. Die Barmänner sollen nicht Gegenseitig auf sich warten müssen. Für einen Barmann, bsp. `Bar1`, dürfen Sie eine *nichtdeterministische Zuweisung*

```
g1  := BIER or LIMO;  // entweder BIER oder LIMO nach g1
```

verwenden.

Füllen Sie für die Kunden und die Barleute den fehlenden *Koordinationskode* aus, um das oben geschilderte Verhalten sicherzustellen. Ihre Lösung soll verklemmungsfrei sein. Begründen Sie, daß Ihre Lösung sich nicht verklemmt.

Aufgabe 3 (Xinu-Prozeßzustände (3 Punkte)) Listen Sie die Xinu-Prozeßzustände auf, erläutern Sie, warum diese eingeführt wurden. Nennen Sie zudem die eventuell assoziierten Warteschlangen und erklären Sie, wie diese logisch organisiert sind.

Aufgabe 4 (Xinu & Uhrinterrupts (4 Punkte)) Nehmen Sie an, ein Uhr-Interrupt wird ausgelöst, der zwei schlafende Prozesse aufweckt. Einer davon habe höhere Priorität als der momentan ausgeführte Prozeß (der "current"-Prozeß). Beschreiben Sie das Aufrufgeschehen. Dies ist z.B. nicht so detailiert nötig das genau aufgeschrieben wird wie die Register gesichert werden.

Aufgabe 5 (Xinu-I/O (5 Punkte)) Beschreiben Sie das Ablaufgeschehen in Xinu, wenn ein auf der Tastatur eingegebenes Zeichen durch den Gerätetreiber Tty auf dem Bildschirm ausgegeben wird. Diskutieren Sie die Fälle `RAW`, `COOKED` und `CBREAK` Mode getrennt. Wer (welcher process, welche Hardware, welche wichtigen Prozeduren, z.B. Assemblerprozeduren müssen nicht erwähnt werden) ist in den verschiedenen Modien bei diesem Ablauf beteiligt?